UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

“LAS PRUEBAS EN EL DESARROLLO DE SOFTWARE”

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

PRESENTA:

CINDY CAMPOS CHIU

DIRECTOR DE TESIS:

ING. ALBERTO TEMPLOS CARBAJAL

CIUDAD UNIVERSITARIA ABRIL 2015.

AGRADECIMIENTOS

Primero, me gustaría agradecer sinceramente a mi asesor de Tesis, Ing. Alberto Templos Carbajal, por su esfuerzo y dedicación. Sus conocimientos, su orientación, su manera de trabajar, su paciencia y su motivación han sido fundamentales para realizar este trabajo.

A mis papás, Antonio Campos y Rosa Chiu, por siempre haber estado junto a mí y para mí en cada momento de mi vida, por su preocupación y su amor. Porque siempre fueron y serán mi motor para realizar las cosas, gracias por corregir mis faltas y celebrar mis triunfos pero sobre todo, por impulsarme a ser mejor cada día.

A mi abuelito, Juan Chiu, por sus enseñanzas, su amor y por depositar toda su confianza en mí.

A mis amigos, pero sobre todo a: Eric, Arturo, Claudia y Carla, por ser una parte importante de mi vida. Gracias por su amistad, apoyo y cariño.

TABLA DE CONTENIDO

1.- Introducción

La calidad del software es un conjunto de procesos bien definidos para lograr la satisfacción de los clientes y usuarios con sistemas que cumplan de la mejor manera su propósito. Existe un proceso dentro de la administración del ciclo de vida de las aplicaciones que en los últimos 10 años ha sido de vital importancia para impulsar la calidad de los sistemas: las pruebas de software.

Actualmente no podemos imaginar la operación y crecimiento de grandes empresas (incluidas empresas de renombre) sin el apoyo y dirección de software desarrollado de forma específica para cubrir las necesidades primordiales de los clientes. El funcionamiento de maquinaria y equipamiento depende en gran medida del software y la calidad en ellos como un sello distintivo para los usuarios.

Las pruebas de software se han convertido en un factor determinante para logar el éxito de sistemas, de ahí la importancia de establecer una metodología que nos guíe el camino para llegar a nuestro objetivo, un software con la mayor calidad posible. Aunando en este tema ¿qué es calidad? y ¿cómo podemos estar seguros que el sistema cubre todos y cada uno de los aspectos que conforman la calidad?, estas preguntas y otros temas relacionados son los que veremos en el contenido de este documento.

Las ventajas de establecer un proceso de calidad son:

- Detectar la mayor cantidad de errores posibles antes de salir a producción
- Ayudar a los administradores a la toma de decisiones
- Buscar los escenarios seguros para el uso del producto
- Evaluar la calidad
- Verificar la corrección del producto
- Asegurar la calidad

El presente es un trabajo en el cual se tratará el tema de pruebas de software y se definirán algunos aspectos importantes para así conocer el aseguramiento de la calidad y el proceso de pruebas que conlleva.

2.- Objetivo

En este documento se explicará de manera clara el papel de las pruebas en las metodologías de desarrollo, se darán a conocer las técnicas, tipos y etapas de las pruebas unificadas en una estrategia para asegurar la calidad del software. A manera de ejemplo se realizará el proceso de pruebas al módulo web “Agenda” con la finalidad de mostrar de manera práctica los elementos mínimos necesarios para determinar si el módulo cumple con los requerimientos definidos.

En el área de sistemas encontramos diferentes roles que contribuyen a la construcción y evaluación del producto, el rol del ingeniero de pruebas es vital para el progreso de todo tipo de instituciones y para el éxito comercial, debido a esto, el ingeniero con formación en sistemas, programación, redes o a fines, se ha visto en la necesidad de desarrollar un perfil como ingeniero de pruebas aprovechando su conocimiento técnico y explotando todas las habilidades adquiridas en su formación educativa, adicional a su conocimiento técnico el ingeniero de pruebas necesita desarrollar habilidades como los que se mencionan a continuación:

- Curioso
- Perceptivo
- Análisis
- Atento a detalles
- Escéptico y con actitud crítica
- Aptitudes para una buena comunicación
- Diplomacia

3.-Fundamentos de pruebas

Objetivo

Proporcionar los elementos teórico-prácticos necesarios para entender, expresar e identificar el entorno de las pruebas de software, como lo es: su propósito, beneficios, características implícitas y explícitas, riesgos y estrategias de prueba.

Algo primordial de las pruebas de software es alcanzar la calidad, pero nos preguntábamos ¿qué es la calidad de software? Según el estándar ISO9126 la calidad de software se define como el grado de satisfacción del usuario con respecto a los requerimientos solicitados contemplando las necesidades implícitas y explicitas del mismo.

Como podemos observar lograr la calidad no es algo sencillo, el ISO 9126 dice que la calidad del software se divide en atributos funcionales y atributos no funcionales, así que para alcanzar la calidad debemos validar lo siguiente:

Atributos funcionales:

- Adecuación
- Exactitud
- Interoperabilidad
- Seguridad
- Cumplimiento de la funcionalidad

Atributos no funcionales de la calidad

- Usabilidad
- Eficiencia
- Mantenibilidad
- Portabilidad
- Fiabilidad

Cuando el sistema evaluado cumple con estas características podemos decir que es un sistema de calidad.

3.1 ¿Qué son las pruebas?

Las pruebas de software son importantes porque aseguran el correcto cumplimiento de la funcionalidad del producto, ayudan a ganar confianza, confirman la fiabilidad del uso y previenen defectos en producción, lo cual tiene un impacto económico positivo en la empresa en cuestión.

Las pruebas de software son una actividad primordial en el proceso de “aseguramiento de la calidad”.

El conjunto de actividades de pruebas dentro del proceso de desarrollo de software, son conocidas como proceso básico de pruebas, el cual incluye:

Planeación. Donde se hace un esquema de ¿qué se va a probar?, ¿cómo se va a probar?, ¿quién lo va a probar? y ¿cuándo? A la salida de esta fase podemos mencionar que se obtiene un plan de pruebas maestro que incluye estrategia y enfoque de pruebas.

Análisis y diseño de pruebas. En esta fase se analizan los requerimientos y se diseñan los casos de prueba, incluyendo en esta actividad casos de prueba positivos y casos de prueba negativos, podemos encontrar más detalle de lo que debe contener un caso de prueba dentro del estándar IEEE 829. A la salida de esta fase se tendrán los casos de prueba.

Ejecución de pruebas. Esta fase es la más importante ya que, es donde los casos de prueba son ejecutados en un ambiente de prueba o calidad para validar que los requerimientos especificados se hayan implementado de la manera correcta, es en este punto donde se aplica la estrategia de pruebas. A la salida de esta fase tendremos el avance de ejecución y los reportes a directivos (generar información para las partes interesadas).

Evaluación de resultados. En esta fase es donde determinamos si se han alcanzado los objetivos de las pruebas, es decir, si la implementación de los requerimientos fue la óptima, lo ideal es que el usuario realice también otras pruebas para dar el visto bueno a la aplicación y ésta pueda ser puesta en producción.

Cierre de pruebas. Es la última fase del proceso de pruebas, aquí archivamos toda la documentación generada y se realiza una carta de aceptación de cierre que debe ser firmada por los directivos involucrados, también es bueno incluir los riesgos activos de la aplicación, si es que aplica.

Estas actividades podrán ser implementadas en cualquier momento de dicho proceso de desarrollo.

De manera general, se puede decir que las pruebas de software permiten determinar si el producto generado satisface las especificaciones establecidas. Así mismo, una prueba de software permite detectar la presencia de errores que pudieran generar salidas o comportamientos inapropiados durante su ejecución.

3.2 Etapas, técnicas y tipos de pruebas

El proceso de pruebas define etapas que especifican en qué momento del desarrollo del software comenzaremos a probar. Cada etapa tiene un objetivo diferente, técnicas que nos ayudan a definir qué y cómo vamos a probar Lo anterior depende de la etapa en la que se comiencen las pruebas. Por último, tenemos tipos de pruebas para validar la funcionalidad, para validar atributos no funcionales e incluso para evaluar un sistema después de haber sufrido cambios, la figura 3.1 describe cómo se desglosan las pruebas:

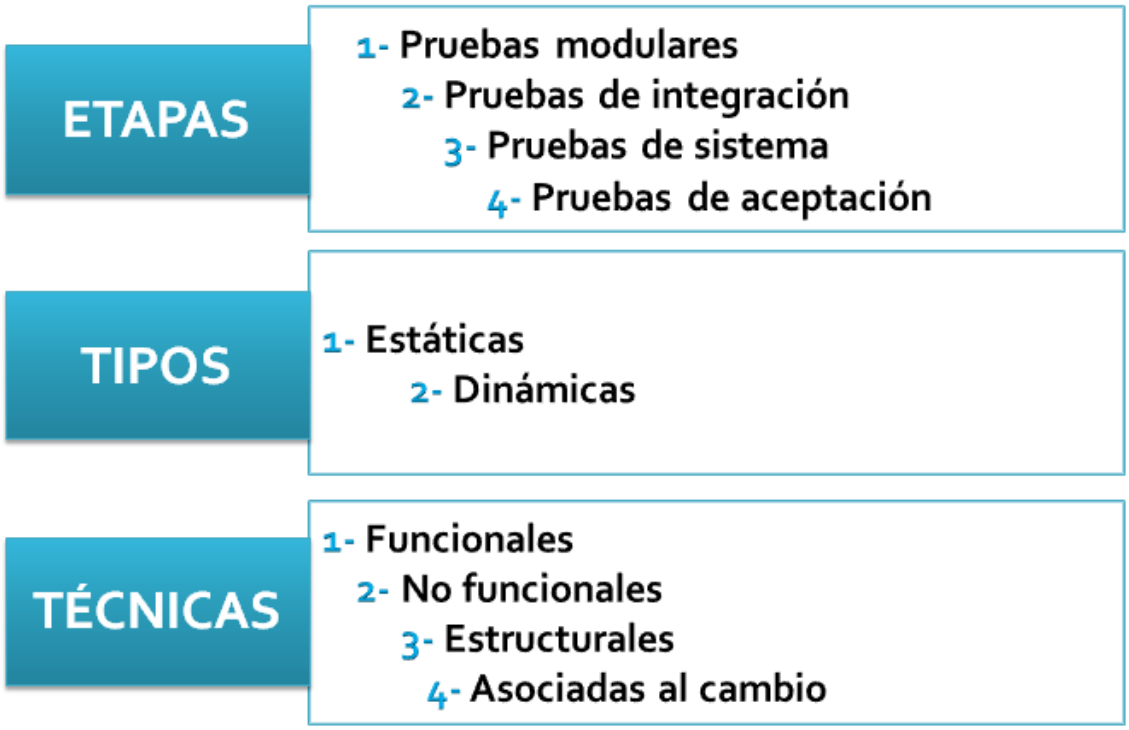


Figura 3.1 Etapas, tipos y técnicas de pruebas

3.3 ¿Por qué probar el software?

Ya hemos hablado de los beneficios de probar software, adentrándonos más en el tema, a continuación se muestran los propósitos más relevantes de por qué probar software:

El propósito de las pruebas es:

- Detectar la mayor cantidad de errores posibles
- Ayudar a los administradores a la toma de decisiones
- Buscar los escenarios seguros para el uso del producto
- Evaluar la calidad
- Verificar la corrección del producto
- Asegurar la calidad
- Competitividad comercial
- Alcanzar mayor calidad en el software
- Cambios en la tecnología
- Reducción de costos y riesgos
- Incremento de productividad

En resumen las pruebas son una inversión, a continuación se mencionan tres puntos importantes por los cuales se debe probar el software.

- ✓ **Mejora de la calidad de un producto software.** El proceso de pruebas ayuda a suministrar o aportar al software los atributos deseados, por ejemplo: retirar defectos que conducen a fallos.

- ✓ **Reducción del riesgo de detectar errores.** Las actividades de pruebas de software adecuadas reducirán el riesgo de encontrar errores durante la fase de operación del software.

- ✓ **Satisfacer compromisos.** La ejecución de pruebas puede ser un requisito obligatorio por parte del cliente, debido a normas legales, así como al cumplimiento de estándares propios de una industria.

El software puesto en producción con defectos puede causar altos costos tanto a usuarios como a proveedores. A continuación se mencionan algunos ejemplos de fallo:

1. La destrucción del Mariner I (1962). 18,5 millones de dólares.

Una sonda espacial que se dirigía a Venus, se desvió de la trayectoria de vuelo prevista poco después del lanzamiento. Desde control se destruyó la sonda a los 293 segundos del despegue. La causa fue una fórmula manuscrita que se programó incorrectamente.

2. La catástrofe del Hartford Coliseum (1978). 70 millones de dólares.

Apenas unas horas después de que miles de aficionados abandonaron el Hartford Coliseum, el techo se derrumbó por el peso de la nieve. La causa: cálculo incorrecto introducido en el software CAD utilizado para diseñar el coliseo.

3. El gusano de Morris (1988). 100 millones de dólares.

El estudiante de posgrado Robert Tappan Morris fue condenado por el primer ataque con “gusanos” a gran escala en Internet. Los costos de limpiar el desastre se cifran en 100 millones de dólares. Morris, es hoy profesor en MIT.

4. Error de cálculo de Intel (1994). 475 millones de dólares.

Un profesor de matemáticas descubrió y difundió que había un fallo en el procesador pentium de Intel. La sustitución de chips costó a Intel 475 millones.

5. Explosión del cohete Arian (1996). 500 millones de dólares.

En el 1996, el cohete Ariane 5 de la Agencia Espacial Europea estalló. El Ariane explotó porque un número real de 64 bits (coma flotante) relacionado con la velocidad se convirtió en un entero de 16 bits.

6. Mars Climate Orbiterm (1999). 655 millones de dólares.

En 1999 los ingenieros de la NASA perdieron el contacto con la Mars Climate Orbiter en su intento que orbitase en Marte. La causa, un programa calculaba la distancia en unidades inglesas (pulgadas, pies y libras), mientras que otro utilizó unidades métricas.

7. El error en los frenos de los Toyota (2010). 3 billones de dólares.

Toyota retiró más de 400.000 de sus vehículos híbridos en 2010, por un problema software, que provocaba un retraso en el sistema anti-bloqueo de frenos. Se estima que entre sustituciones y demandas el error le costó a Toyota 3 billones de dólares.

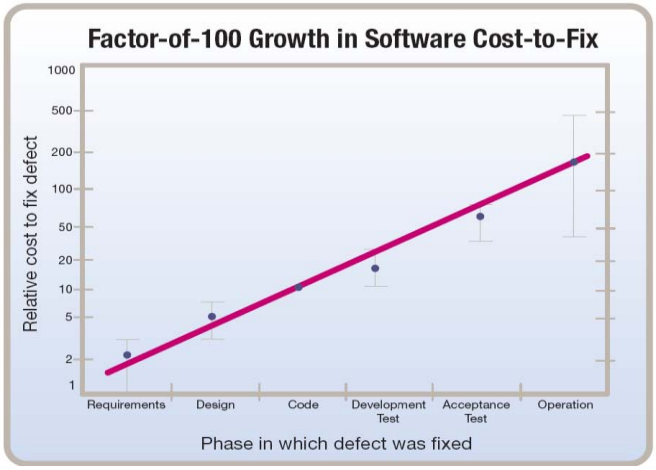
8. Las migraciones por el año 2000. 296,7 billones de dólares.

Se esperaba que el bug Y2K paralizase al mundo a la medianoche del 1 de enero 2000, ya que mucho software no había sido previsto para trabajar con el año 2000. El mundo no se acabó, pero se estima que se gastaron 296,7 billones de dólares para mitigar los daños.

9. En el año 2000 hubo una sobredosis radiológica en el Instituto Nacional del Cáncer de Panamá, los ingenieros de la empresa Multidata Systems International calcularon erróneamente la dosis de radiación que un paciente podría recibir durante la terapia de radiología. El fallo estaba en el software de control de la máquina de rayos, lo que provocó que al menos ocho pacientes murieran por las altas dosis recibidas y otros 20 tuvieran problemas de salud graves.

El costo de los defectos

- Los costos de eliminar defectos se incrementan con el tiempo durante el cual el defecto permanece en el sistema.
- La detección de errores en etapas tempranas permite la corrección de los mismos a costos reducidos.



Gráfica 3.1 Costo de los defectos

3.4 Requerimientos de pruebas

En el aseguramiento de la calidad la toma de requerimientos claros y precisos es indispensable para logar los objetivos.

Definición de requerimiento. Una condición o necesidad de un usuario para resolver un problema o alcanzar un objetivo. Una condición o capacidad que debe estar presente en un sistema o componentes de sistema para satisfacer un contrato, estándar, especificación u otro documento formal.

Se debe recordar que existen requerimientos explícitos, que son los que el usuario define y exige de manera clara y precisa, pero también se tienen los requerimientos implícitos que aunque el usuario no los especifique espera que el sistema sea entregado con éstos, por ejemplo: que el sistema sea bonito, que el sistema sea rápido, que el sistema sea entendible, etcétera.

Un requisito describe un atributo funcional deseado o considerado obligatorio.

Las actividades para identificar los requerimientos se muestran en la figura 3.2.

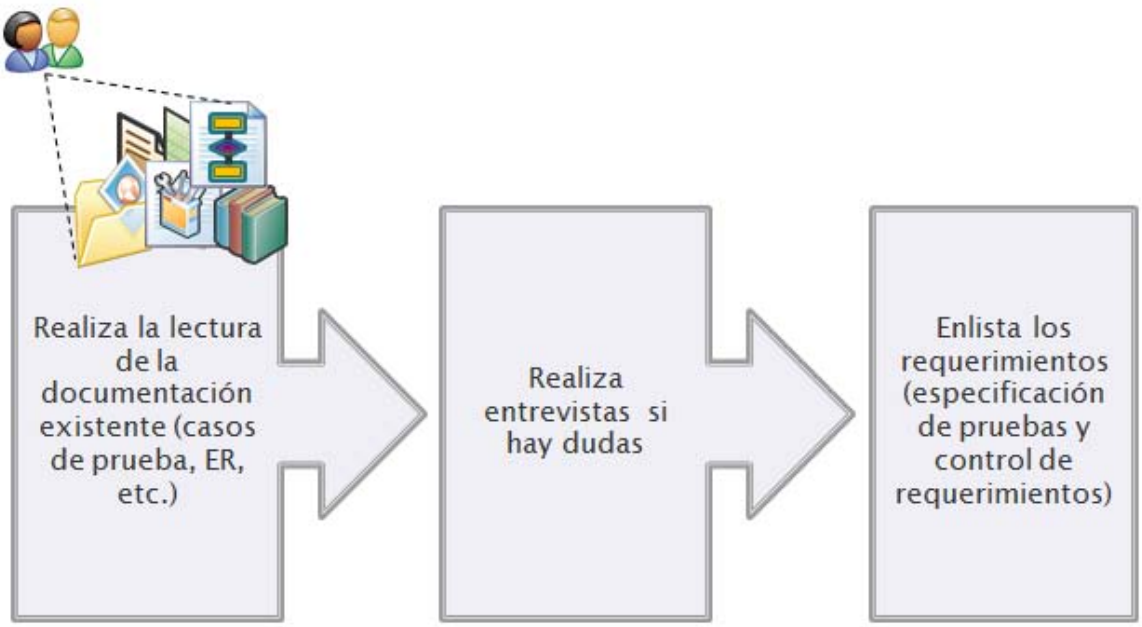


Figura 3.2 Levantamiento de requerimientos

Las consideraciones para identificar requerimientos se muestran en la figura 3.3

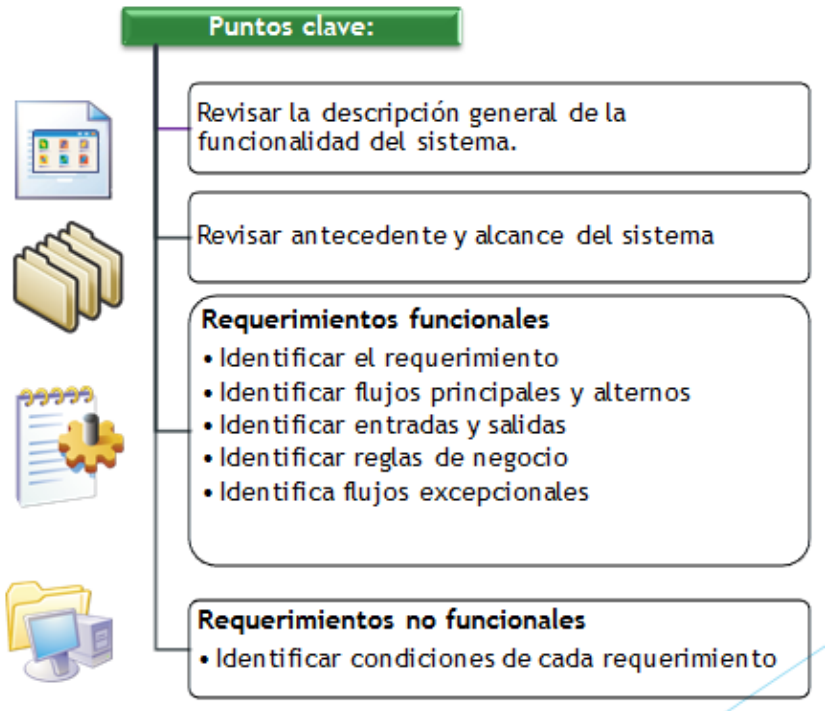


Figura 3.3 Identificación de requerimientos

3.5 Caso de prueba

Anteriormente se mencionó que en la etapa de análisis y diseño se elaboran los casos de prueba, pero ¿qué es un caso de prueba?

Definición: documentación que especifica las acciones a seguir para llegar a un objetivo específico (resultado esperado) que contiene o debería contener datos de entrada, resultados esperados y un conjunto de condiciones de ejecución de un elemento de prueba.

Lo que debe contener un caso de prueba según el estándar IEEE 829, es:

- Precondiciones
- Valores de entrada
- Resultados esperados
- Postcondiciones
- Identificador único
- Dependencia de otros casos de prueba
- Requisitos
- Forma en la cual se debe ejecutar el caso de prueba
- Prioridad

Tipos de casos

- **Casos de prueba positivos (Test To Pass):** Son aquéllos que se diseñan para probar el requerimiento tal cual se diseñó y construyó. Estos casos de prueba aseguran que las funciones descritas en los casos de uso o requerimientos funcionales realmente estén implementadas en el sistema o módulo que se esté probando y que funcionen como se definieron. Muestran funcionalidad.

- **Casos de prueba negativos (Test To Fail):** Son diseñados con la intención de romper el requerimiento, módulo o sistema a probar, es decir, comprueban situaciones en las que hay tratamientos con errores, por ejemplo: datos de entrada inválidos.

- **Happy path:** Escenario default de un requerimiento. No se consideran caminos alternos, excepcionales o condiciones de error.

Los casos de prueba se pueden crear de manera formal e informal a partir de requisitos.

Si la funcionalidad especificada es el objetivo de las pruebas, los métodos utilizados se denominan métodos basados en la especificación (caja negra).

Si la estructura interna de un objeto es investigada, los métodos utilizados se denominan métodos basados en la estructura (caja blanca).

3.6 ¿Qué es un bug?

En el proceso de pruebas se registran los problemas encontrados o las incidencias detectadas, esto es lo que un ingeniero de pruebas conoce como *bug*. Dicho de manera técnica un desperfecto en un componente o sistema que puede causar que el sistema falle en desempeñar las funciones requeridas, por ejemplo: una sentencia o una definición de datos incorrecta.

Los defectos son la manifestación física de un error, un error es definido como: acción humana que produce un resultado incorrecto, por ejemplo: un error de programación.

¿Qué causa errores? han escuchado hablar de errar es de humanos. Es cierto, cuando un sistema falla es porque el humano que lo ha programado o creado ha cometido un error, los errores normalmente son causados por:

- Plazos de trabajo excesivos
- Presiones de Tiempo
- Distracciones
- Mala interpretación de los requerimientos

3.7 Testware

Es el medio de trabajo para la aplicación de pruebas de software, como documentación, scripts, resultados esperados, configuración, procedimientos, archivos, bases de datos, y cualquier software adicional o servicios utilizados en pruebas.

3.8 Ingeniero de pruebas

Persona que asegura que el equipo de desarrollo ofrece la calidad necesaria a sus clientes. Centrándose en la entrega de un producto que proporcione más valor del negocio.

Características de un buen probador:

- Curioso
- Atento a los detalles
- No creer todo lo dicho por los desarrolladores
- No debe de temer al hecho de que se puedan detectar defectos de importancia que pudieran tener un impacto sobre la evolución del proyecto
- Aptitud para la comunicación

4. Etapas o niveles de pruebas

4.1 Verificación y validación

En la actualidad existen muchos modelos de desarrollo de software, es un reto para el ingeniero de pruebas amoldarse a cada modelo con el fin de cubrir todas las tareas y actividades de manera óptima. El modelo tradicional de desarrollo y pruebas es conocido como el modelo V, donde se describen las actividades de desarrollo y las correspondientes actividades en pruebas.

El modelo V (figura 4.1) plantea que en todo este proceso se debe llevar a cabo la verificación y validación y se realiza como se menciona a continuación.

Cada nivel de desarrollo se verifica con el nivel anterior, es decir, se comprueba si los requisitos y definiciones de niveles previos han sido implementados de forma correcta.

La validación se refiere a la corrección de cada nivel de desarrollo, de esta manera se comprueba lo adecuado de los resultados de un nivel de desarrollo.

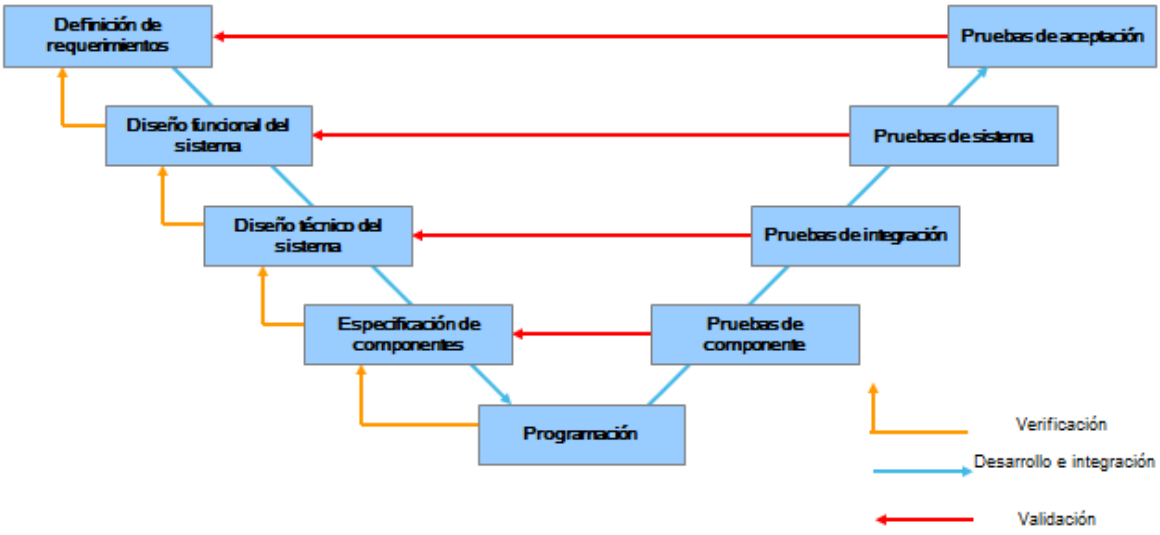


Figura 4.1 Modelo V

De esta manera podemos observar que el proceso de pruebas comienza antes que la ejecución de las mismas, tan pronto comienza el desarrollo se puede comenzar la preparación de las pruebas correspondientes, como es el caso de la revisión de documentos.

4.2 Pruebas unitarias

Como se observa en el modelo V se describen los niveles de prueba existentes, comenzaremos explicando el primer nivel “pruebas unitarias”.

Un componente es la unidad más pequeña especificada de un sistema, las pruebas se llevan a cabo tras la construcción o realización de cada componente para verificar que la implementación se esté llevando conforme a los estándares acordados.

De acuerdo a cada lenguaje de programación se puede hacer referencia a un componente como:

- Prueba de módulo (en C)
- Prueba de clase (en Java o C++)
- Prueba de unidad (en Pascal)

Se hace referencia a los componentes como módulos, clases o unidades, debido a que los desarrolladores pueden estar involucrados en la ejecución de pruebas, pueden llamarse también pruebas de desarrollador.

Es importante mencionar que estas pruebas las ejecuta el desarrollador verificando que su código cumple con lo solicitado y no ha violado ningún estándar que ponga en riesgo la estabilidad del sistema.

Las pruebas de componente podrán comprobar características funcionales y no funcionales de un sistema.

4.3 Pruebas integrales

Las pruebas integrales también son conocidas como pruebas de interfaz, ya que comprueban la interacción entre componentes.

Las pruebas de integración asumen que los módulos ya han sido probados de manera individual (pruebas unitarias).

Implican una progresión ordenada de pruebas que van desde los componentes o módulos y que culminan en el sistema completo.

El orden de integración elegido afecta a diversos factores como los siguientes:

- La forma de preparar casos
- Las herramientas necesarias
- El orden de codificar y probar los módulos
- El costo de la depuración
- El costo de preparación de casos

Existen tres enfoques para realizar pruebas de integración, todos obedecen a modelos de desarrollos incrementales o iterativos, como *scrum, xtreme programing* y cascada, estos enfoques pueden ser:

Ascendente. Cuando el desarrollo del sistema comienza con piezas unitarias y crece hasta formar módulos.

Descendente. Cuando se muestra el *front* de la aplicación pero está hueca por dentro y se comienza a trabajar desde los módulos más grandes hasta llegar al detalle o a las partes unitarias.

***Big Bang*.** Que va de la mano con *xtreme programing* porque el desarrollo no tiene un orden específico y puede ser al azar.

Cualquier enfoque distinto a estos es llamado *ad-hoc* que encaja con el modelo de desarrollo tradicional V.

4.4 Pruebas de sistema

Las pruebas de sistema se llevan a cabo cuando todo el desarrollo ha sido culminado y tenemos una versión preliminar del sistema que saldrá a producción. Esta etapa de pruebas consiste en probar un sistema integrado con el objeto de comprobar el cumplimiento de requisitos especificados.

Las pruebas se hacen con un enfoque desde el punto de vista del usuario.

Las pruebas de sistema se desarrollan utilizando casos de prueba funcionales y no funcionales. Las pruebas funcionales confirman que los requisitos para un uso específico previsto han sido cumplidos (validación).

Las pruebas de sistema no funcionales verifican los atributos de calidad no funcionales, lo vimos en el estándar ISO 9126, con los atributos funcionales y no funcionales de la calidad.

4.5 Pruebas de aceptación

El objetivo en este nivel de prueba es obtener el visto bueno del cliente, no se deberían encontrar defectos funcionales graves en el sistema. Es por ello que las pruebas de aceptación son realizadas por el usuario.

Se puede decir que las pruebas de aceptación son las pruebas de sistema por parte del cliente.

Existen dos tipos de pruebas de aceptación:

- **Pruebas Alfa.** El cliente utiliza el software para hacer el tratamiento de sus procesos de negocio en las dependencias del proveedor.
- **Pruebas Beta.** Estas se ejecutan en las dependencias del cliente.

Las pruebas de aceptación son consideradas como la fase final del proceso para crear una confianza en que el producto es apropiado para su uso, de esta manera se verificará que el software satisface los requisitos del cliente.

5. Tipos de pruebas

5.1 Funcionales

Tal y como su nombre lo indica las pruebas funcionales se enfocan en validar la correcta implementación de las necesidades del cliente. La funcionalidad puede ser vinculada a los datos de entrada y de salida. Los datos de entrada serán ejecutados y mostrarán un resultado y dicho resultado será comparado con el resultado esperado (comportamiento), este proceso se muestra en la figura 5.1.

Figura 5.1 Pruebas funcionales

5.2 No funcionales

Las pruebas no funcionales revisan las características implícitas del sistema, lo que las hace difícil de validar.

Se enfocan a las características de un software, existen pruebas no funcionales que miden atributos que no podrían ser validados sin el uso de una herramienta, a continuación se describen este tipo de pruebas:

Carga. Pruebas a un sistema cubriendo la demanda esperada.

Rendimiento. Rapidez con la cual un sistema ejecuta una determinada función.

Estrés. Someter a la aplicación a una carga mucho mayor a la esperada y evaluar la capacidad del sistema de seguir su operación después de este tipo de demanda.

Existen muchas otras como: seguridad, estabilidad, volumen, robustez.

Otros aspectos no funcionales de calidad:

Fiabilidad. El sistema debe ser confiable a los usuarios para que puedan operar sin preocuparse de sus datos o de la mala operación de un sistema.

Mantenibilidad. El código debe cumplir con ciertas normas y estándares para evitar que se encuentre de forma compleja, el código siempre debe estar comentado para cada función realizada.

Portabilidad. Se deben asegurar las características del sistema; en qué sistema operativo operará, cuál será el navegador de preferencia, etcétera.

5.3 Estructurales

Las pruebas estructurales se enfocan a revisar el código y su estructura.

Es un enfoque de caja blanca, y se mide el grado en el cual la estructura del objeto de prueba ha sido cubierta por los casos de prueba.

No es más que asegurarnos que todo el código es ejecutado con los casos de prueba diseñados, de esa manera se evita que existan ramas muertas o código inaccesible.

5.4 Asociadas al cambio

El objetivo de estas pruebas es validar que la funcionalidad que ya se ha validado siga igual después de una modificación al código, hablamos de re-test, y pruebas de regresión, es decir, probar después de algún cambio.

Probar el sistema después de los cambios. Después de que un objeto de prueba se modificó, los resultados de las pruebas resultan inválidos, por lo que las pruebas deben ser repetidas.

Razones por las cuales el software puede ser modificado:

- Corrección de errores (re-test)
- Extensión funcional (pruebas de regresión)

Prueba de regresión. Repetir una prueba de funcionalidad que ha sido verificada previamente.

Repetición de pruebas. Pruebas tras corrección de errores.

En la mayoría de los casos, las pruebas de regresión completas no son viables debido a sus altos costos y duración, se recomienda que las pruebas de regresión consideren como máximo un 35% del ciclo completo de pruebas.

Es por esto que se deben tomar en cuenta algunos criterios para la selección de casos de prueba de regresión:

- Casos de prueba de prioridad alta
- Probar funcionalidad estándar
- Probar configuración utilizada con más frecuencia
- Probar los flujos de negocio más importantes
- Probar sólo el *happy path*

6. Técnicas de pruebas

6.1 Estáticas

Las técnicas estáticas de pruebas comprenden métodos donde no se ejecutan los componentes u objetos de prueba (sistema).

Las pruebas estáticas incluyen: revisiones y análisis estáticos. Estas pruebas complementan los métodos dinámicos. De esta manera se pueden detectar causas de fallos en lugar de fallos.

La detección temprana de errores ahorra costos.

Ventajas

- Costos más bajos
- Los defectos en la documentación son detectados y corregidos de manera temprana
- Los documentos de alta calidad mejoran el proceso de desarrollo

Desventajas

- Pueden presentarse situaciones de tensión con el autor
- Inversión considerable de tiempo (10% - 15% del presupuesto total)

6.1.1 Revisiones formales e informales

En todo proceso de desarrollo deben existir revisiones para asegurarnos que todo el equipo está en el mismo contexto y no haya opiniones encontradas. El estándar IEEE 1028 nos da la base para llevar a cabo revisiones de manera óptima:

Fases de una revisión. Las fases para llevar a cabo una revisión consisten en la planificación llevada por el líder quien convoca a una reunión, revisión donde expertos en el tema estudian el objeto puesto a prueba, reunión como tal, reconstrucción y seguimiento.

Los roles involucrados son: líder del proyecto, revisores, autor, escriba y moderador.

Los tipos de revisiones son: revisión inspección, donde participan todos los roles y se lleva a cabo en todas las fases; revisión guiada, el autor pide sugerencias sobre su documento; revisión técnica, expertos en la materia dan una solución unánime al problema o defecto detectado y por último la revisión informal que también es conocida como revisión entre pares.

6.1.2 Análisis estático

Este análisis se lleva a cabo sobre el código fuente, consiste en analizar un objeto de prueba sin llevar a cabo la ejecución del mismo.

Herramientas utilizadas para realizar el análisis:

Compilador

- Detecta errores de sintaxis en el código fuente de un programa
- Comprueba la consistencia entre los tipos de variables
- Detecta variables no declaradas y código muerto

Analizador

- Métricas de complejidad
- Estándares
- Acoplamiento de objetos

Análisis del flujo de control

El objetivo es detectar los defectos causados por un desarrollo anómalo del código fuente como se muestra en la figura 6.1. Por ejemplo: código muerto.

Métodos

- La estructura del código se representa con un diagrama de control de flujo
- Grafos dirigidos

  Los nodos representan sentencias
  Las aristas representan la transferencia del flujo de control (bucles, decisiones)

Resultados

- Visión del conjunto del código del programa
- Las inconsistencias pueden ser detectadas con más facilidad, por ejemplo: bucles abandonados por saltos, ramas muertas

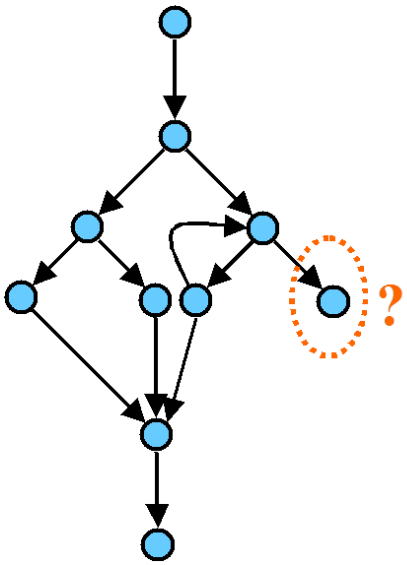


Diagrama de flujo 6.1

6.2 Dinámicas

6.2.1 Caja blanca

El objetivo es analizar el objeto de prueba, ejecutando el objeto de prueba. Se llevan a cabo por medio de revisiones y herramientas.

- Complementan los métodos estáticos
- Costos más bajos, la corrección de errores sucede en etapas tempranas.

Prevención de defectos

Las técnicas de caja blanca, examinan la parte interna del programa, siempre se está observando el código, y los casos de prueba están basados en la estructura interna del programa (figura 6.2). Por ello, la implementación de estas pruebas depende de la disponibilidad del código fuente. Este tipo de pruebas permiten generar casos para ejercitar y validar los caminos de cada módulo, las condiciones lógicas, los bucles, etcétera.

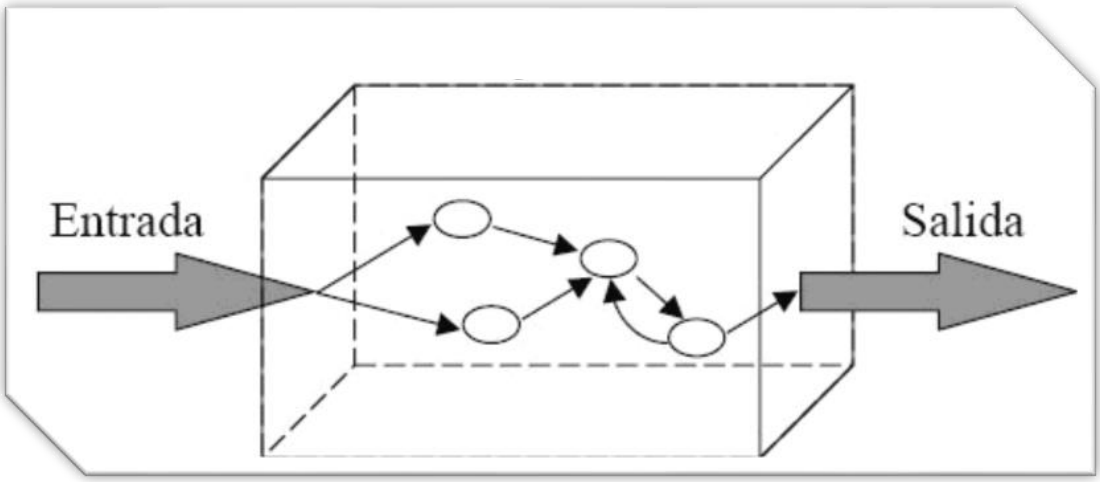


Figura 6.1 Caja blanca

Principales tipos de cobertura

Cobertura de sentencia (puede detectar código muerto, nodos, etcétera), su objetivo es tener el número mínimo de casos de prueba para lograr tocar todos los nodos al menos una vez.

La base de este tipo de cobertura es el diagrama del flujo de control. Todas las instrucciones están representadas por nodos y el flujo de control entre instrucciones está representado por una arista.

$$\text{Cobertura de sentencia}\,(C_0) = \frac{\text{Número de sentencias ejecutadas}}{\text{Número total de sentencias}} * 100\%$$

Ejemplo:

Se tiene el siguiente segmento de código:

```
If (i>0) {
        If (j>10) {
                For (k=i; k>10; k--) {
                }
        }
}
```

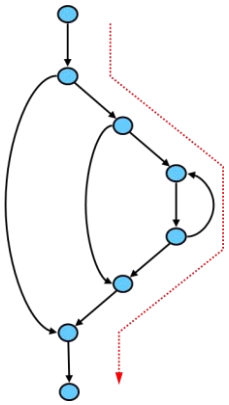


Diagrama de flujo 6.2

Análisis

Considerando el diagrama de flujo 6.2:

Se tienen dos sentencias '*if*' y un bucle '*for*' dentro del segundo '*if*'

Existen tres caminos diferentes para este segmento de programa:

La primera sentencia '*if*' permite dos direcciones o caminos.

La dirección de la derecha de la primer sentencia '*if*' se divide nuevamente a partir del segundo '*if*'.

Todas las sentencias de este programa pueden ser alcanzadas con el camino de la derecha.

Por lo tanto, un sólo caso de prueba es suficiente para alcanzar el 100% de cobertura de sentencia.

En conclusión, este tipo de pruebas sirve para detectar código muerto, o bien, código que nunca es ejecutado.

Si existe código muerto en el programa no se podrá lograr una cobertura de sentencia al 100%.

Por otro lado, no pueden ser detectadas instrucciones faltantes.

Cobertura de decisión o rama (todas las aristas deben ser cubiertas por lo menos una vez), su objetivo al igual que las otras técnicas es lograr una cobertura de decisión al 100%, esto se logra con el número mínimo de casos de prueba que toquen al menos una vez todas las aristas o ramas.

Cobertura de decisión también se conoce como cobertura de rama.

$$\text{Cobertura de decisión } (C_1) = \frac{\text{Número de decisiones ejecutadas}}{\text{Número total de decisiones}} * 100\%$$

O bien:

$$\text{Cobertura de rama}(C_1) = \frac{\text{Número de ramas cubiertas}}{\text{Número total de ramas}} * 100\%$$

Ejemplo:

Se tiene el siguiente diagrama que representa el segmento de un programa (diagrama de flujo 6.2).

Hay tres caminos diferentes que conducen a través del diagrama.

La primer sentencia tiene dos direcciones (‘*if*’).

Siguiendo el camino de la derecha del primer ‘*if*’, se tiene nuevamente otra sentencia ‘*if*’ en el cual se encuentra un *loop.*

Se necesitan tres casos de prueba para alcanzar una cobertura de decisión del 100%.

Si se utilizan solamente las dos direcciones de la derecha tenemos que:

$$\text{Cobertura de decisión}(C_1) = \frac{9}{10} * 100\% = 90\%$$

Para lograr una cobertura de decisión del 100% son necesarios el mismo número de casos que para la cobertura de sentencia.

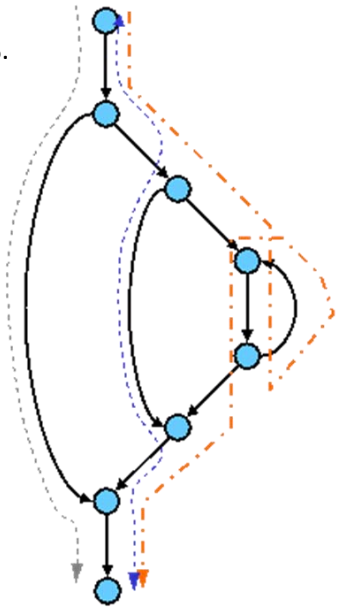


Diagrama de flujo 6.2

En conclusión, una cobertura de decisión del 100% requiere, al menos los mismos casos de prueba que la cobertura de sentencia.

Por tanto, una cobertura de decisión del 100% siempre incluye una cobertura de sentencia del 100%.

Con este tipo de prueba no se pueden detectar sentencias faltantes.

Cobertura de condición (detectar defectos resultantes de la implementación de condiciones múltiples: *OR, AND* y *XOR*, muchas veces cuando tenemos una condición compuesta no sabemos cómo validarla, la cobertura de condición nos menciona que debemos realizar todas las posibles combinaciones y resultados derivados de combinación de la condición compuesta.

Existen 3 tipos de cobertura: mínima cobertura de condición, múltiple cobertura de condición, y mínima múltiple cobertura de condición.

Cobertura de camino (ejecución de todos los posibles caminos a través de un programa), el objetivo es lograr el 100% de cobertura de camino que consiste en tomar en cuenta todos los posibles caminos con respecto a los bucles y las sentencias, se debe considerar un camino que entre al bucle si es que aplica y otro que no entre al bucle.

Para lograr el objetivo de esta prueba y así alcanzar un porcentaje definido de cobertura tenemos que:

$$\text{Cobertura de camino} = \frac{\text{Número de caminos cubiertos}}{\text{Número total de caminos}} * 100\%$$

Ejemplo:

Se tiene el siguiente diagrama, el cual es un segmento de programa (diagrama de flujo 6.3).

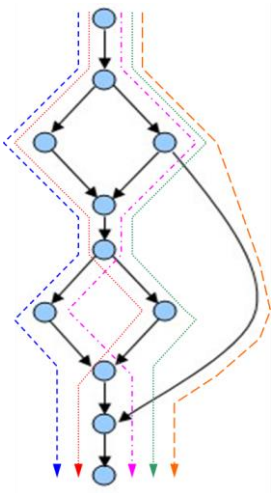


Diagrama de flujo 6.3

Contiene tres sentencias '*if*'.

Tres caminos distintos conducen a través del diagrama, lo cual logra una cobertura de decisión completa.

Sin embargo, pueden ser ejecutados cinco caminos distintos.

Por tanto, son necesarios cinco casos de prueba para lograr un 100% de cobertura de camino, dos para un 100% de sentencia y tres casos para un 100% de cobertura de decisión.

Todas las partes de un programa deben ser ejecutadas por lo menos una vez, es necesario el uso de herramientas. Generalmente se aplica en pruebas de componente o de integración.

Es importante mencionar que las pruebas de caja blanca identifican el origen de un defecto, más no la manifestación del mismo.

6.2.2 Caja negra

La técnica de caja negra, se enfoca en probar el sistema sin tomar en cuenta la estructura interna del mismo, su objetivo es validar que las salidas sean las esperadas (figura 6.2).

Se centra en encontrar las circunstancias en las que el sistema no se comporta conforme a las especificaciones establecidas.

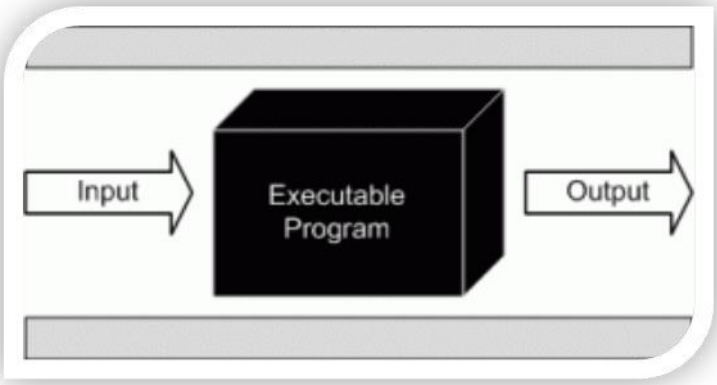


Figura 6.2 Caja negra

Las técnicas más comunes de caja negra son:

Partición de equivalencia (posibles valores divididos en clases, valores de entrada y valores de salida). Se agrupan todos los valores para los cuales se espera que el programa tenga un comportamiento común

(rango de valores), y esa es una clase de equivalencia, existen clases de equivalencia válidas y clases de equivalencias inválidas.

Valores límite. Complementan a la partición equivalente, se debe prestar mucha atención en que los límites deben estar correctamente definidos y programados.

Transición de estado. La transición de estados nos enuncia que todo sistema se mueve por transiciones de un paso a otro, nos podemos guiar por las transacciones válidas y las transiciones inválidas.

Tablas de decisión. Estas pruebas consideran que para encontrar el resultado esperado deben estar en conjunto varias condiciones que son los detonadores del resultado, llamado como causa y efecto.

Ejemplo:

Se tiene el siguiente formato de registro

Número De Empleado	Nombre	Meses Trabajando	Directivo

Dónde:

Número de empleado es un campo de tres dígitos enteros positivos, sin incluir el 000.

Nombre es un campo alfanumérico de máximo 30 caracteres.

Meses trabajando es un campo que indica el número de meses que lleva el empleado en la empresa. Entero positivo de 3 dígitos, incluyendo el 000.

Directivo es un campo de un solo carácter que puede ser "+" para indicar que el empleado es un directivo o "-" para indicar que no lo es.

De acuerdo a lo anterior, tenemos que:

Condición	Clases válidas	Clases inválidas
Número de empleado	-Número de tres dígitos, enteros positivos $000 > x \leq 999$	-Número menor a tres dígitos -Número mayor a tres dígitos -x = 000 -Números negativos -Números decimales -Caracteres no numéricos -Cadena nula

Nombre	-Cadena alfanumérica de máximo 30 caracteres	-Cadena mayor a 30 caracteres -Cadena nula
Meses trabajando	-Número de tres dígitos, enteros positivos $000 \geq x \leq 999$	-Número menor a tres dígitos -Número mayor a tres dígitos -Números negativos -Números decimales -Caracteres no numéricos -Cadena nula
Directivo	-Cadena de 1 carácter x = + x = -	-Cadena nula -Cadena de más de 1 carácter -Carácter diferente a "+" o "-"

7. Metodologías

Para procesos de desarrollo basados en la metodología *RUP* o métodos tradicionales, implementar una metodología de pruebas es totalmente viable, teniendo en cuenta que estas metodologías están orientadas a la documentación y a la formalización de todas las actividades ejecutadas.

Un proceso de pruebas formal está compuesto, cuando menos por las siguientes 5 típicas etapas:

- Planeación de pruebas
- Diseño de pruebas
- Implementación de pruebas
- Evaluación de criterios de salida
- Cierre del proceso

Planeación de pruebas

En esta etapa se llevan a cabo las primeras actividades correspondientes al proceso de pruebas, tiene como resultado un entregable denominado plan de pruebas el cual debe estar conformado mínimo por los siguientes aspectos:

Alcance de la prueba. Determina qué funcionalidades del producto (software) serán probadas durante el proceso de pruebas.

Tipos de pruebas. En este punto se determina qué tipos de pruebas requiere el producto. No todos los productos de software requieren de las mismas pruebas o la aplicación de todas ellas. Los posibles tipos de prueba que se pueden aplicar son: pruebas de stress, pruebas de rendimiento, pruebas de carga, pruebas funcionales, pruebas de usabilidad, pruebas de regresión, entre otros.

Estrategia de pruebas. **T**eniendo en cuenta que no es viable probar con base a todas las posibles combinaciones de datos, es necesario determinar a través de un análisis de riesgos sobre que funcionalidades debemos centrar nuestra atención. Adicionalmente, una buena estrategia de pruebas debe indicar los niveles de pruebas (ciclos) que aplicaremos y la intensidad o profundidad a aplicar para cada nivel de pruebas definido. En este punto también es importante definir los criterios de entrada y salida para cada ciclo de pruebas a ejecutar.

Criterios de salida. Entre las partes involucradas en el proceso, se define de manera formal, bajo qué condiciones se puede considerar que una actividad de pruebas fue finalizada. Los criterios de salida se deben definir para cada nivel de pruebas a ejecutar. Algunos ejemplos de criterios de salida que pueden ser utilizados son: porcentaje de funcionalidades de alto riesgo probadas con éxito, número defectos críticos y/o mayores aceptados, etcétera.

Otros aspectos. Tal y como se realiza en cualquier plan de proyecto, se debe incluir una estimación de tiempos, los roles y/o recursos que serán parte del proceso, la preparación del entorno de pruebas, etcétera.

Diseño de pruebas

Una vez elaborado y aprobado el plan de pruebas, el equipo de trabajo debe iniciar el análisis de la documentación existente del sistema, con el objeto de iniciar el diseño de los casos de prueba. El diseño de los casos debe considerar la elaboración de casos positivos y negativos.

Implementación y ejecución de pruebas

La ejecución de pruebas debe iniciar con la creación de los datos de prueba necesarios para ejecutar los casos de prueba diseñados. La ejecución de estos casos, puede realizarse de manera manual o automatizada; en cualquiera de los casos, cuando se detecte un fallo en el sistema, este debe ser documentado y registrado en una herramienta que permita gestionar los defectos. Una vez que el defecto ha sido corregido en su respectivo proceso de depuración, es necesario realizar un re-test para confirmar que el defecto fue solucionado de manera exitosa. Por último, es indispensable ejecutar un ciclo de regresión que permita asegurar que los defectos corregidos en el proceso de depuración no hayan desencadenado otros defectos en el sistema.

Evaluación de criterios de salida

Los criterios de salida son necesarios para determinar si es posible dar por finalizado un ciclo de pruebas. Para esto, es conveniente definir una serie de métricas que permitirán al finalizar un proceso de pruebas, comparar los resultados obtenidos contra las métricas definidas, sí los resultados obtenidos no superan las métricas definidas, no es posible continuar con el siguiente ciclo de pruebas.

Cierre del proceso

Durante este período de cierre, el cual históricamente se ha comprobado que se le destina muy poco tiempo en la planeación, se deben cerrar las incidencias reportadas, se debe verificar si los entregables planeados han sido entregados y aprobados, se deben finalizar y aprobar los documentos de soporte de prueba, analizar las lecciones aprendidas para aplicar en futuros proyectos, etcétera.

8. El papel del ingeniero en computación en las pruebas de software

El rol del ingeniero de pruebas es vital para el progreso de todo tipo de instituciones, debido a esto el ingeniero con formación en sistemas, programación, redes, etcétera. Se ha visto en la necesidad de desarrollar un perfil como tester aprovechando su conocimiento técnico y explotando todas las habilidades adquiridas en su formación educativa.

Las características primordiales de un tester son:

- Curioso, perceptivo, atento a los detalles. No todos los detalles se ponen de manifiesto
- Con el objeto de **comprende**r los escenarios prácticos del **cliente**
- Con el objeto de poder analizar la estructura de la prueba
- Con el objeto de **descubrir detalles** de dónde se pueden manifestar fallos
- Escéptico y con actitud crítica
- Los objetos de prueba contienen defectos, usted sólo debe encontrarlos
- No creer todo lo dicho por los desarrolladores

No se debe temer al hecho de que se pudieran detectar defectos de importancia que pudieran tener un impacto sobre la evolución del proyecto

- Aptitudes para la comunicación
- Necesarias para llevar **malas noticias** a los desarrolladores
- Necesarias para vencer estados de frustración
- Tanto cuestiones técnicas como prácticas, relativas al uso del sistema, deben ser entendidas y comunicadas
- Una comunicación positiva puede ayudar a evitar o facilitar situaciones difíciles
- Para establecer una relación de trabajo con los desarrolladores a corto plazo
- Experiencia, factores personales influyen en la ocurrencia de errores

El ingeniero de pruebas ha adaptado su perfil para enfocarse al aseguramiento de la calidad, aplicando sus conocimientos para validar de manera consciente la funcionalidad del software. Debido a su ingenio y experiencia colabora con la creación de herramientas para facilitar el proceso de pruebas. Los expertos técnicos dependiendo del área colaboran con las decisiones y mejoras de una implementación.

9. Caso de estudio

CERTIFICACIÓN DE APLICACIÓN WEB “AGENDA”
Definición del proceso de pruebas

Contenido

- Propósito
- Contexto actual
- Objetivo
- Estrategia de pruebas
- Alcance
- Criterios de suspensión y reanudación de pruebas
- Requerimientos de prueba
- Diseño de casos de prueba
    - Pruebas funcionales
    - Pruebas en la etapa de sistema
    - Técnica de pruebas caja negra
        - Partición equivalente
        - Valores límite
        - Casos de uso
- Resumen de ejecución
- Defectos detectados
- Evaluación de riesgos
- Evaluación y cierre
- Liberación a producción
- Conclusiones

9.1 Propósito

Esta sección del documento engloba los siguientes objetivos:

1. Definir el alcance y enfoque de las pruebas a realizar
2. Identificar los componentes de software que deben ser probados y las características o escenarios específicos a ser probados
3. Delimitar las tareas de pruebas y el esfuerzo requerido para desarrollar cada una de ellas
4. Identificar los recursos necesarios (humanos, software, hardware, etcétera) para cubrir y desarrollar las tareas de pruebas
5. Describir los riesgos que pueden afectar el plan de pruebas
6. Establecer los artefactos de pruebas que se producirán durante el proceso de pruebas

9.2 Contexto actual

En el local host de una máquina cualquiera se tiene acceso a una agenda web la cual permite almacenar datos de contactos, solicitando el ingreso de los siguientes campos:

- Nombre (alfabético de hasta 15 caracteres)
- Apellido (alfabético de hasta 15 caracteres)
- Correo electrónico (Alfanumérico de hasta 15 caracteres con validación de @ y dominio)
- Número de celular a 10 dígitos (numérico de 10 dígitos)
- Edad (numérico de 2 dígitos entero positivo)

9.3 Objetivo

Realizar el proceso de pruebas al módulo web “agenda” con la finalidad de contar con los elementos mínimos necesarios para poder determinar si el módulo cumple con la definición de calidad según la definición de ISO 9126:

Calidad de software:

“La totalidad de la funcionalidad y prestaciones de un producto de software que están relacionadas con su capacidad de satisfacer las necesidades explícitas o implícitas”.

9.4 Estrategia de pruebas

Según la necesidad comercial y del usuario es básico para el correcto funcionamiento del módulo validar:

- Adecuación
- Exactitud

- Interoperabilidad
- Seguridad
- Cumplimiento de funcionalidad

Se contempla que la aplicación local soporta hasta 1000 contactos, por tal motivo no es necesario ni será caso de estudio para este ejemplo realizar pruebas de volumen o estrés, nos enfocaremos solamente a validar los atributos funcionales de la calidad y algunos atributos no funcionales como el look & feel.

A continuación se muestra la estrategia de pruebas definida dentro del alcance.

9.5 Alcance

En este apartado se mostrará cómo llevar a cabo el proceso de pruebas básico para el módulo web "Agenda".

Las pruebas comenzarán con la etapa de las pruebas Modulares, en las cuales mediante pruebas dinámicas, casos de prueba diseñados y set de datos válidos e inválidos, se verificara la funcionalidad de cada módulo a probar, durante esta etapa se identificarán los casos de prueba adecuados para ejecutar la siguiente etapa que son las pruebas integrales las cuales tienen como objetivo validar la integración de la aplicación de un módulo a otro hasta concluir el flujo de trabajo exitoso (happy path).

- **Pruebas Modulares**. Pruebas que validan la correcta funcionalidad del Módulo o Subsistema
- **Pruebas de Integración**. Pruebas que aseguran el correcto funcionamiento del flujo de trabajo

Las funcionalidades a probar son las siguientes:

- Agregar contacto
- Consultar todos los contactos
- Editar contacto
- Eliminar contacto
- Buscar contacto

Las pruebas se enfocarán a validar la funcionalidad para el aseguramiento de la calidad del software según el ISO 9126.

Las pruebas se realizarán a nivel de sistema, es decir no existirán stubs ni drivers además de que la funcionalidad a validar deberá estar completa.

Las técnicas existentes para la validación de un sistema se componen de técnicas estáticas y dinámicas, siendo estas últimas las más utilizadas debido su fácil aplicación y sus óptimos resultados; las técnicas dinámicas están divididas en técnicas de caja negra y caja blanca, éstas últimas se enfocan a validar la

complejidad y cobertura del código; para este caso de estudio sólo aplicaremos las técnicas dinámicas de caja negra:

- Partición equivalente
- Valores límite
- Casos de uso

El tipo de pruebas no funcionales, pruebas estructurales y pruebas asociadas al cambio, serán objeto de estudio en un planteamiento futuro quedando fuera de alcance para este proyecto.

9.6 Criterios de suspensión y reanudación de pruebas

La ejecución de pruebas será suspendida en los siguientes casos:

- No se cuenta con un ambiente de pruebas
- La versión entregada no supera la validación general, pues presenta muchas fallas
- El incremento de defectos es muy alto
- No se cuenta con datos de prueba

Se podrán reanudar las pruebas una vez corregidos o solucionados los puntos anteriores.

9.7 Requerimientos de prueba

ID	Requerimiento de prueba	Tipo	Complejidad
3	Agregar contacto	Funcionalidad	Alto
5	Editar contacto	Funcionalidad	Medio
7	Eliminar contacto	Funcionalidad	Medio
8	Buscar contacto	Funcionalidad	Medio
9	Consultar todos los contactos	Funcionalidad	Simple
42	Look & feel	Funcionalidad	Simple

9.8 Diseño de casos de prueba

9.8.1 Pruebas funcionales

Las pruebas funcionales se dividen en estáticas y dinámicas, la revisión está enfocada a las pruebas funcionales-dinámicas de caja negra, entendiendo que nuestro objeto de estudio serán únicamente las entradas y salidas en el sistema sin importar cómo opera internamente.

9.8.2 Pruebas en la etapa de sistema

De acuerdo a la estrategia de pruebas, los casos son ejecutados a nivel sistema, es decir una versión completa sin stubs o drivers, una versión lo más semejante al ambiente productivo.

Se aplicarán pruebas dinámicas de caja negra, las cuales comprenderán de las siguientes técnicas:

Técnica de caja negra

- Partición equivalente
- Valores límite
- Casos de uso

9.8.3 Técnica de pruebas caja negra

a) Partición equivalente

La partición en clases de equivalencia se realiza dividiendo los posibles valores de entrada y los posibles valores de salida. El rango de valores definido se agrupa en clases de equivalencia.

Para este caso tenemos 5 campos con las siguientes características

- Nombre (alfabético de hasta 15 caracteres)
- Apellido (alfabético de hasta 15 caracteres)
- Correo electrónico (Alfanumérico de hasta 15 caracteres con validación de @ y dominio)
- Número de celular a 10 dígitos (numérico de 10 dígitos)
- Edad (numérico de 2 dígitos entero positivo)

La partición equivalente será aplicada de la siguiente manera:

Nombre (alfabético de hasta 15 caracteres)
Toda cadena alfabética con un máximo de hasta 15 caracteres es válida, ejemplo: • Clase de equivalencia válida: Johannes, Josafat, Heriberto, Citlalli • Clase de equivalencia inválida: Klauss12, K@rla, Rumpelstiltskinn, Nadia_360 • Clase de equivalencia inválida valores numéricos: 12564878, 1315488, -469841521 Se considera que para cada clase de equivalencia definida sólo es necesario un representante, es decir no es práctico probar con todos los valores ya que el resultado siempre será el mismo.

Apellido (alfabético de hasta 15 caracteres)
Toda cadena alfabética con un máximo de hasta 15 caracteres es válida, ejemplo: • Clase de equivalencia válida: Villegas, Zambrano, Rio, Castillo • Clase de equivalencia inválida caracteres especiales: corona_&, c@rranza, castañeda# • Clase de equivalencia inválida valores numéricos: 12564878, 1315488, -469841521 Se considera que para cada clase de equivalencia definida sólo es necesario un representante, es decir no es práctico probar con todos los valores ya que el resultado siempre será el mismo.

Correo electrónico (alfanumérico de hasta 15 caracteres con validación de “@” y dominio)
Toda cadena alfanumérica con un máximo de hasta 15 caracteres es válida seguida por un @ y un dominio, ejemplo: • Clase de equivalencia válida: 12@you.com, dante@bett.com, m_1245_hjgt@adam.com • Clase de equivalencia inválida: corona_12@, c@rranza@p.com, castañeda#12_23, 12345678, alberto_lopez Se considera que para cada clase de equivalencia definida sólo es necesario un representante, es decir no

es práctico probar con todos los valores ya que el resultado siempre será el mismo.

Número de celular a 10 dígitos (numérico de 10 dígitos)
Toda cadena numérica de 10 dígitos enteros positivos Ejemplo: • Clase de equivalencia válida: 5555555555, 2345678952, 5544228877, 01234567890 • Clase de equivalencia inválida cadena mayor a 10 dígitos: 55442288771245, 012345678901245 • Clase de equivalencia inválida CADENA MENOR A 10 DIGITOS: 552266, 457896, 254546446 • Clase de equivalencia inválida valores no numéricos : roberto23, “#$%%12, 56****Clau • Clase inválida números negativos: -5555555555, -1234567890 • Clase inválida números decimales: 5.098783125, 1234.567891 Se considera que para cada clase de equivalencia definida sólo es necesario un representante, es decir no es práctico probar con todos los valores ya que el resultado siempre será el mismo.

Edad (numérico de 2 dígitos, entero positivo)
Toda cadena numérica de 2 dígitos como máximo (0-99), ejemplo: • Clase de equivalencia válida: 1, 56, 98, 88 • Clase de equivalencia inválida cadena mayor a 2 dígitos: 554, 012345678901245, 12345 • Clase de equivalencia inválida cadena menor a 0 : -552266, -457896, -254546446, -0.12 • Clase de equivalencia inválida números con decimales: 1.5, 18.3, 99.68 • Clase de equivalencia inválida valores no numéricos: roberto23, “#$%%12, 56****Clau Se considera que para cada clase de equivalencia definida sólo es necesario un representante, es decir no es práctico probar con todos los valores ya que el resultado siempre será el mismo.

b) Valores límite

El análisis de valores límite amplía la técnica de partición en clases de equivalencia introduciendo una regla para seleccionar representantes.

Los valores frontera (valores límite) de la clase de equivalencia deben ser probados de forma intensiva.

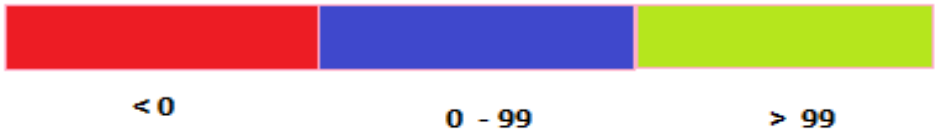
¡La experiencia demuestra que, con mucha frecuencia, los errores tienen lugar en los límites del rango de valores!

Valores límite

Se aplica la técnica de valor límite al campo “Edad”

Edad (numérico de 2 dígitos, entero positivo)

Toda cadena numérica de 2 dígitos (0-99), ejemplo:



Valores límite inferiores: -1, 0

Valores límite superiores: 99,100

c) Casos de Uso

Identificador	Nombre
CU001	Agregar contacto
Requerimiento de prueba	
REQ003	
Precondiciones	
• NA	
Objetivo	
• Permitir al usuario crear un contacto dentro de la agenta web	
Flujo Principal	
1. El caso de uso inicia cuando el usuario ejecuta la aplicación agenda en su pc. 2. El módulo presenta formulario para crear nuevo contacto, campo para realizar búsqueda y opción para consulta de todos los contactos. 3. El usuario ingresa los campos: nombre, apellido, correo electrónico, edad y número telefónico y da clic en “Guardar”. 4. Si la información es correcta el módulo manda el mensaje **M-001**. 5. El contacto se muestra en la lista de contactos. 6. Fin de caso de uso.	
Flujo Alterno CU001-1	
1. El flujo alterno inicia cuando el usuario ejecuta la aplicación agenda en su pc. 2. El módulo presenta formulario para crear nuevo contacto, campo para realizar búsqueda y opción para consulta de todos los contactos. 3. El usuario ingresa el campo nombre, apellido, correo electrónico, edad y número telefónico y da clic en guardar (al ingresar el usuario deja campos en blanco o en un formato incorrecto). 4. Si la información es incorrecta el módulo manda el mensaje **M-002** o **M-003** colocando el foco en el campo donde se presente el error. 5. Fin del flujo alterno.	
Flujo Alterno CU001-2	
1. El flujo alterno inicia cuando el usuario ejecuta la aplicación agenda en su pc. 2. El módulo presenta formulario para crear nuevo contacto, campo para realizar búsqueda y opción para consulta de todos los contactos. 3. El usuario ingresa el campo nombre, apellido, correo electrónico, edad y número telefónico y da clic en el botón “Cancelar”. 4. El módulo limpia los campos y no guarda la información. 5. Fin del flujo alterno.	

Postcondiciones
- El contacto se guarda en la agenda web.
Anexos
- Matriz de Mensajes

Id	Tipo de Mensaje	Botón Relacionado	Mensaje
M-001	Confirmación	Guardar	Contacto guardado
M-002	Error	Guardar	Existen caracteres no válidos
M-003	Error	Guardar	Campo requerido

Identificador	Nombre
CU002	Consultar todos los contactos
Requerimiento de prueba	
REQ009	
Precondiciones	
• NA	
Objetivo	
• Permitir al usuario consultar los contactos registrados en la agenda web	
Flujo Principal	
1. El caso de uso inicia cuando el usuario ejecuta la aplicación agenda en su pc. 2. El módulo presenta formulario para crear nuevo contacto, campo para realizar búsqueda y opción para consulta de todos los contactos. 3. El usuario da clic en el botón “Todos los contactos”. 4. El módulo muestra la información de los contactos existentes con opción a “Editar” y “Eliminar”. 5. Fin de caso de uso.	
Flujo Alterno CU002-1	
• NA	
Postcondiciones	

- Los contactos existentes se muestran
Anexos
- NA

Identificador	Nombre
CU003	Editar contacto
Requerimiento de prueba	
REQ005	
Precondiciones	
• Debe haber contactos existentes	
Objetivo	
• Permitir al usuario modificar contactos	
Flujo Principal	
1. El caso de uso inicia cuando el usuario ejecuta la aplicación agenda en su pc. 2. El módulo presenta formulario para crear nuevo contacto, campo para realizar búsqueda y opción para consulta de todos los contactos. 3. El usuario da clic en el botón “Todos los contactos”. 4. El módulo muestra la información de los contactos existentes con opción a “Editar” y “Eliminar”. 5. El usuario da clic en el botón “Editar” del contacto que desee modificar. 6. El módulo presenta la información del contacto habilitada para su edición. 7. El usuario realiza cambios en alguno(s) de los campos: nombre, apellido, correo electrónico, edad y/o teléfono y da clic en “Actualizar”. 8. El módulo muestra mensaje **M-004** y muestra el contacto con la información actualizada. 9. Fin de caso de uso.	
Flujo Alterno CU003-1	
1. El flujo alterno inicia cuando el usuario ejecuta la aplicación agenda en su pc. 2. El módulo presenta formulario para crear nuevo contacto, campo para realizar búsqueda y opción para consulta de todos los contactos. 3. El usuario da clic en el botón “Todos los contactos”. 4. El módulo muestra la información de los contactos existentes con opción a “Editar” y “Eliminar”. 5. El usuario da clic en el botón “Editar” del contacto que desee modificar. 6. El módulo presenta la información del contacto habilitada para su edición. 7. El usuario realiza cambios en alguno(s) de los campos: nombre, apellido, correo electrónico, edad y/o número telefónico y da clic en “Cancelar”. 8. El módulo no guarda los cambios. 9. El módulo muestra pantalla todos los contactos.	

10. Fin del flujo alterno.

Postcondiciones

- El contacto se actualiza

Anexos

- Matriz de Mensajes

Id	Tipo de Mensaje	Botón Relacionado	Mensaje
M-004	Confirmación	Actualizar	Contacto actualizado

Identificador	Nombre
CU004	Eliminar contacto

Requerimiento de prueba

REQ007

Precondiciones

- Debe haber contactos existentes

Objetivo

- Permitir al usuario borrar contactos

Flujo Principal

1. El caso de uso inicia cuando el usuario ejecuta la aplicación agenda en su pc.
2. El módulo presenta formulario para crear nuevo contacto, campo para realizar búsqueda y opción para consulta de todos los contactos.
3. El usuario da clic en el botón “Todos los contactos”.
4. El módulo muestra la información de los contactos existentes con opción a “Editar” y “Eliminar”.
5. El usuario da clic en el botón “Eliminar” del contacto que desee borrar.
6. El módulo muestra mensaje **M-005**.
7. El usuario confirma el mensaje.
8. El módulo muestra mensaje **M-006**.
9. Si el contacto fue eliminado de manera correcta ya no se muestra en la lista.
10. Fin de caso de uso.

Flujo Alterno CU004-1

1. El flujo alterno inicia cuando el usuario ejecuta la aplicación agenda en su pc.
2. El módulo presenta formulario para crear nuevo contacto, campo para realizar búsqueda y opción para consulta de todos los contactos.
3. El usuario da clic en el botón “Todos los contactos”.
4. El módulo muestra la información de los contactos existentes con opción a “Editar” y “Eliminar”.
5. El usuario da clic en el botón “Eliminar” del contacto que desee borrar.
6. El módulo muestra mensaje **M-005**.
7. El usuario da clic en “Cancelar”.
8. El módulo no elimina el contacto.
9. Fin del flujo alterno.

Postcondiciones

- El contacto es eliminado

Anexos

- Matriz de Mensajes

Id	Tipo de Mensaje	Botón Relacionado	Mensaje
M-005	Confirmación	Eliminar	¿Desea eliminar contacto?
M-006	Confirmación	Aceptar	Contacto eliminado

Identificador	Nombre
CU005	Buscar Contacto

Requerimiento de prueba

REQ008

Precondiciones

- NA

Objetivo

- Permitir al usuario realizar búsqueda de contactos

Flujo Principal

1. El caso de uso inicia cuando el usuario ejecuta la aplicación agenda en su pc.
2. El módulo presenta formulario para crear nuevo contacto, campo para realizar búsqueda y opción para consulta de todos los contactos.
3. El usuario ingresa el texto a buscar en el campo “Buscar contactos” y da clic en el botón “Buscar”.

4. El módulo muestra los contactos que coinciden con el texto ingresado. 5. En caso de no existir coincidencias el módulo no muestra información. 6. El usuario Fin de caso de uso.
Flujo Alterno CU004-1
• NA
Postcondiciones
- Se muestran resultados de la búsqueda en caso de existir.
Anexos
- NA

9.9 Resumen de ejecución

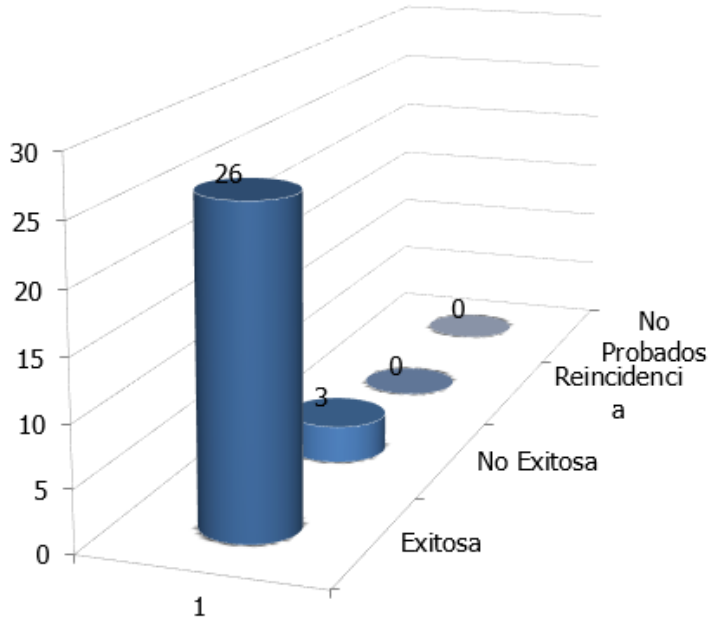
ID	Requerimiento	ID	Tipo	Caso de prueba	Resultado
3	Agregar contacto	50	TTF	ClaseInválida.NúmerosNegativos.Teléfono	Pass
		26	TTF	CamposRequeridos	Pass
		16	TTF	ClaseInválida.CaracteresEspecialesYNuméricos.Apellido	Pass
		14	TTF	ClaseInválida.CaracteresEspecialesYNuméricos.Nombre	Pass
		22	TTF	ClaseInválida.CaracteresNoNuméricos.Edad	Pass
		21	TTF	ClaseInválida.CaracteresNoNuméricos.Teléfono	Pass
		18	TTF	ClaseInválida.CorreoElectrónico	Pass
		23	TTF	ClaseInválida.NumérosDecimales.Edad	Pass
		49	TTF	ClaseInválida.NumérosDecimales.Teléfono	Pass
		19	TTF	ClaseInválida.X<10Dígitos.Teléfono	Pass
		20	TTF	ClaseInválida.X>10Dígitos.Teléfono	Pass
		17	TTF	ClaseInválida.X>15Caracteres.Apellido	Pass
		15	TTF	ClaseInválida.X>15Caracteres.Nombre	Pass
		44	TTF	ContactoDuplicado	Pass
		48	TTF	ValidarCorreoElectrónico	Pass
		25	TTF	ValoresLímiteInválidos.Edad	Pass
		28	TTP	Cancelar	Pass
		13	TTP	ClasesVálidas	Pass
		24	TTP	ValoresLímiteVálidos.Edad	Pass
		39	TTF	ClaseInválida.X>15Caracteres.CorreoElectrónico	Pass
5	Editar contacto	27	TTP	EditarContacto	Pass
		29	TTP	EditarContacto - Cancelar	Pass
7	Eliminar contacto	30	TTP	EliminarContacto	Pass
		31	TTP	EliminarContacto - Cancelar	Pass
8	Buscar contacto	33	TTF	BuscarContactoNoExistente	Pass
		32	TTP	BuscarContactoExistente	Pass
9	Consultar todos los contactos	34	TTP	ConsultarContactos	Pass
42	Look & feel	36	TTP	L&F – PáginaPrincipal	Fail
		37	TTP	L&F – TodosLosContactos	Fail

Estatus de prueba

Ejecución

Estatus	Ciclo 1
Exitosa	26
No Exitosa	3
Reincidencia	0
No Probados	0

General	
Requerimientos de prueb.	6
Casos de Prueba	29
Happy Path	1
Automatizados	0



9.10 Defectos detectados

ID	Defecto	Riesgo	Consecuencia	Impacto	Estatus
45	Homogenizar etiquetas	Ninguno	NA	NA	Cerrado
46	Cabeceras	Ninguno	NA	NA	Cerrado
47	Etiqueta buscar contacto	Ninguno	NA	NA	Cerrado

9.11 Evaluación y cierre

Evaluación. El módulo sometido a pruebas presentó un comportamiento continuo sin riegos de operación elevados, el equipo de aseguramiento de calidad recomienda instalar el aplicativo en producción con un período de estabilización y mejora continua.

Cierre. Se ejecutaron 29 casos de prueba en un ciclo completo. En el cual se detectaron 3 defectos no funcionales. Tras la corrección de los defectos se ejecutó el re-test en el cual los defectos reportados fueron corregidos correctamente. Completando la valoración del módulo se aplicaron pruebas de regresión del flujo principal, en el cual no se detectaron defectos.

9.12 Liberación a producción

El aplicativo será liberado a producción con la valoración del área Aseguramiento de la Calidad, adicional a esto, es requerida la autorización por escrito del:

- Gerente de sistemas
- Líder de Pruebas
- Gerente del área de negocio

9.13 Comentarios finales

La operación del sistema es estable, los riesgos probables son menores ya que no interfieren ni comprometen la funcionalidad principal del sistema, adicional a esto no hay comprometidos bienes materiales ni daño físico.

El sistema entrará en un proceso de mejora continua.

10. Conclusiones

Las pruebas de software son todo un proceso, dentro del proceso de desarrollo, que debe llevarse a cabo para asegurar la calidad del software. Algo que debemos reconocer es la naturaleza compleja de las mismas. Las pruebas pueden iniciarse en cualquier momento dependiendo del tipo de pruebas que se requiera.

El proceso de pruebas es de vital importancia dentro del ciclo de vida del software, de esta manera se puede verificar la calidad del producto antes de su puesta en producción.

El proceso de pruebas ayudó a la recolección de los requerimientos, así como a evitar inconsistencias, omisiones y confusiones. El análisis permitió que las correcciones se realizaran de manera más fácil y en etapas tempranas.

Como todo aspecto de la vida, la calidad es sinónimo de satisfacción y seguridad. Las pruebas de software ganan cada día más terreno ya que lo que siempre buscamos es mantener a nuestro cliente contento. En este aspecto existe un tema de complicidad con el usuario ya que es el único que puede dar el visto bueno del trabajo realizado por todo un equipo, desde los analistas hasta el ingeniero de pruebas.

El ingeniero de pruebas debe aprender a desarrollar una buena comunicación con todo el equipo para que éste acepte las críticas u observaciones de manera positiva, ésta no es una tarea fácil incluso es un punto relevante para el éxito del proyecto.

Tener una estrategia de pruebas en la planeación, desarrollo y entrega del producto puede reducir costos y/o evitar gastos en corrección de incidentes que son detectados en etapas tempranas y de esa forma asegurar la calidad y reducir la probabilidad de riesgos que se puedan presentar en el desarrollo del software y su puesta en producción.

Al final, las pruebas terminan siendo una inversión y pueden ahorrar tiempo y dinero.

11. Mesografía y bibliografía

- Guía de certificación del ISTQB® nivel básico
- http://standards.ieee.org/

  10.01.14
- http://www.fiuxy.com/noticias/1667647-20-errores-informaticos-excelente.html

  27.01.14
- Ingeniería del software.

  Séptima edición. Ian Sommerville

  Pearson educación. S. A., Madrid 2005
- Ingeniería del software. Un enfoque práctico.

  Séptima edición. Roger S. Pressman, Ph. D.

  McGraw Hill, 2010